

# **SYSTEM AND METHOD FOR USING CHARACTER SET MATCHING TO ENHANCE PRINT QUALITY**

## **Cross-Reference to Related Applications**

This application claims priority from, and incorporates by reference, U.S. Provisional application serial number 60/114,574 filed December 31, 1998. The subject matter of this application is related to the subject matter of copending U.S. Patent application serial numbers \_\_\_\_\_, entitled "System and Method For Outputting Character Sets In Best Available Fonts," "System and Method For Output Of Multipart Documents," and "System and Method For Highlighting Of MultiFont Documents," respectively, each having the same inventor as this application and each being assigned or under obligation of assignment to the same assignee as this application, each filed the same day as this application, and each incorporated by reference. The subject matter of this application is also related to the subject matter of copending U.S. Patent application serial numbers \_\_\_\_\_, entitled "System and Method For Evaluating Character Sets," "System and Method For Evaluating Character sets To Determine A Best Match Encoding A Message," "System and Method For Evaluating Character Sets Of A Message Containing A Plurality Of Character Sets," and "System and Method For Evaluating Character Sets To Generate A Search Index," respectively, each filed the same day as this application and each being assigned or under obligation of assignment to the same assignee as this application, and each also incorporated by reference.

## **Field of the Invention**

The invention relates to the field of communications, and more particularly to the management of character set information within documents so that the most appropriate font may be used to output documents in printed form which have an arbitrary language origin.

### **Background of the Invention**

With the use of the Internet, email and related electronic services, communications software has been increasingly called upon to handle data in a variety of formats. While the barriers to simple communications have been removed from many hardware implementations, the problem of software being unable to display text in different languages remains.

For instance, a person browsing the World Wide Web may wish to input a search string in their native language. Some Web pages or search engines will simply accept that string in the form in which it was input, but not process the spelling, syntax or character set in native form. The search engine then performs a search as though the search were in English, usually resulting in no hits. Other Web pages may allow user to manually specify the desired language for browsing and searching.

To display multilingual data correctly on the output side when a document is located and ready to be outputted on a computer screen, sent to a printer or otherwise using the right font is a common problem that developers encounter when writing international applications, including Web applications.

In the pursuit of better and more uniform multilingual documents, the International Standards Organization (ISO) and other bodies have developed a universal character set standard referred to as Unicode, Version 2.0 of which was released in 1996. The current Unicode standard is a 16-bit protocol encoding 25 different scripts as well as at least 38,885 separate characters. Scripts are in general higher-level collections of related characters from a character set which may be assembled for use in one or more languages.

Ultimately, displaying Unicode data becomes a problem of dynamically selecting the closest font available on a system to best express a subject document. The commercial TrueType™ and OpenType™ font sets can only output the appropriate glyphs or symbols for a subset of Unicode ranges. It is therefore necessary to generate the right character set (charset) font flag information, that is, to identify the font best matching the data to be displayed when creating the graphical font object for display or printing.

Unfortunately, documents often originate from unknown sources and unknown languages and even if the original text is encoded in Unicode, it can be difficult to identify in which character set or Unicode target range the content has been encoded with. Once the character set is identified, it can be mapped to the corresponding font flag information used to create the logical font under a given operating system. For example, text that can be encoded as Code page 874 would be mapped to THAI\_CHARSET in Microsoft™ Windows™.

The display of Unicode data consequently relates to a process of selecting a font and using whatever system application programming interface (API) is available to output the text, a process which becomes even more complicated when the text is in multiple languages. Because the selected font might not be able to render all the characters from all the different character sets used in multipart, multilanguage documents, that type of data must be broken into different textual segments that use the same character set and display them separately using the appropriate font.

Therefore, among other technical complexities, the software developer must solve at least two fundamental problems when trying to accommodate multilingual output. The first is to determine which character set or Unicode ranges the text has been encoded with. The second, to choose the font that will be able to render the

characters most correctly. Other problems exist, including the selection of most appropriate fonts for developing printed output.

### **Summary of the Invention**

The invention overcoming these and other problems in the art relates to a system and method for character set matching to achieve best printed output, in which character set information is embedded in fonts and large-capacity fonts containing symbols applicable to a variety of languages include font tag information representing their capabilities. Application software, operating system software and other resources may therefore interrogate the font tag information to determine whether a given font may represent the characters of a message on a character-by-character basis. A linked list of matching fonts is built, which may be traversed to output the document to a screen display, printer or elsewhere. Textual documents having more than one language may be analyzed in successive runs, in which the most suitable font is selected for each segment for printing such as on a laser, inkjet or other printing apparatus.

### **Brief Description of the Drawings**

The invention will be described with respect to the accompanying drawings, in which like numbers represent like elements.

Fig. 1 illustrates an overall system for character set communication according to the invention.

Fig. 2 illustrates the encoding layout for the Unicode standard.

Fig. 3 illustrates the relationship between big fonts, characters, scripts and other entities according to the invention.

Fig. 4 illustrates the operation of an enumerator module according to the invention for generating a link list of fonts.

Fig. 5 illustrates the operation of a font manager module according to the invention.

Fig. 6 illustrates the relationship between a big font object and Unicode encoding.

Fig. 7 illustrates a multipart document for output according to the invention.

Fig. 8 illustrates a text run object for separating individual portions of a multiple language document for different font expression according to the invention.

Fig. 9 illustrates an embodiment of the invention implemented in a Lotus Notes™/Domino™ environment.

Fig. 10 illustrates the transmission of a message over a network according to the invention.

### **Detailed Description of the Illustrative Embodiments**

First, one illustrative environment in which the invention operates will be described, in which documents to be processed are assumed to be stored in the Unicode format. Unicode is a 16-bit universal character encoding that encompasses virtually all characters commonly used in the languages of today's world. Unicode encodes text by scripts, and not necessarily by individual languages.

The Unicode standard assigns the same code point to characters or ideographs that are shared among multiple languages, even though these characters may have different meanings and pronunciations in those different languages. For this reason, one character can be rendered by more than one glyph or symbol from different character sets, and this can potentially add logical errors to the displayed text.

Scripts can be defined as a higher-order collection of related characters which can be assembled to form one or more languages. For example, Cyrillic script can be used for the expression of Russian, Ukrainian, Bulgarian and other languages. Hebrew script can be used to express Modern Hebrew, Biblical Hebrew, and other languages. Each script has its own properties and rules,

including logical order (right to left, left to right, neutral), dynamic composition, ligature and other attributes.

Ultimately, it is the responsibility of the font resources to implement the rules and behaviors proper to each script as outlined by the Unicode standard. For a font to be Unicode conformant, it must correctly interpret and render a subset of Unicode characters, providing support for one or many specific scripts and languages. To avoid ambiguity, the invention helps to transparently identify the language or character set being used in the data so that a best match to available output fonts may be made.

Otherwise, in the absence of the invention the operating system would have to select whatever default language is available in the given script. This can result in character rendering errors. Therefore, the invention operates to choose the optimal font for display as well as to select the correct script in the font, and may query the font properties and display the text output according to correct rules of language. Moreover, the invention may process texts composed in several different languages, one after the other for output. An illustration of the overall text processing according to the invention is shown in Fig. 10.

The Unicode standard uses different ranges for each of its constituent scripts, as illustrated in Fig. 2. Given a character code point within the 16-bit address range of Unicode, it is straightforward using Unicode's encoding layout to identify the corresponding script by location. But because a single script can be used by more than one language, it is only by looking at the surrounding text and using statistical methods that the character set of the original language can be determined with any degree of certainty.

It should be noted that ASCII characters are used in nearly all scripts and therefore do not reveal significant information about the original language.

Conversely, the presence of idiosyncratic characters which are strongly associated with particular languages can be used to infer the use of that language in the original text. For example, Hiragana characters in a stream of Han ideographs is a strong indication that the text is Japanese, and a stream of Hangul characters is almost certainly Korean. Other scripts have known associations with various particular languages.

In general, in terms of character set evaluation or identification the invention uses a character table bank against which the ability of a number of character sets to encode a given character is tested. When a message of unknown origin is presented to the system, its characters are parsed and tested against the character table bank to identify which of the pool of character sets can express each character.

A character set which contains a match for every character of the message is likely to be the native encoding of the original message. Tallies of matches to individual characters across all available character sets in the character table bank can also be made for the message as a whole. An overall architecture of a character set analysis system is illustrated in Fig. 1.

The invention in one regard uses statistical methods to provide an automatic and rigorous language evaluation facility by which the text represented in Unicode is tested against a bank of available language character sets, in order to determine which or any of those candidate character sets can express the text in its entirety. The invention evaluates which character sets are capable of expressing the text from the language bank, to present to a user or otherwise.

The invention may assign a rating to those character sets that can express the given message, in order to determine which of the character sets is the most appropriate to use to express the message. The invention may likewise evaluate which character set permits searching and reading of text expressions, improving

the quality of search results, all as more fully described in the aforementioned copending U.S. application serial numbers \_\_\_\_\_.

As illustrated in Fig. 3, fonts which are capable of representing characters from multiple scripts are referred to as big fonts 502. Within a big font 502, scripts are identified by font tag 504 that encodes information about the capabilities of that font, including an indication of scripts which employ the characters of that font. In one embodiment the font tag may comprise a unique 4-byte identifier, but other formats are contemplated by the invention. Furthermore, as illustrated in Fig. 6, in the particular environment of True Type™ fonts, big fonts 502 may be tagged with a font signature 506 that contains information about the Unicode subranges currently supported.

For example, Microsoft Windows™ uses the following structure to exchange font signature information.

Table 1

```

Typedef struct tagFONTSIGNATURE
{
    DWORD fsUsb[4];           //Unicode subranges

    DWORD fsCsb[2];          //Windows and OEM code pages

} FONTSIGNATURE;
  
```

Individual applications can query big fonts about their multilingual capabilities by accessing the font tag 504 according to the invention. More specifically, client software can inquire whether a given font may be able to render the text of a current document correctly. Microsoft™ Windows™ provides the



following function to obtain the font signature of a font currently selected in a Device context, as will be understood by persons skilled in the art.

Table 2

```
int WINAPI GetTextCharsetInfo(HDC hdc,

LPFONTSIGNATURE lpSig,

DWORD

dwFlags);
```

The procedure call in Table 2 above can only be used, however, to validate a given font for matching a character set which is already specified. To dynamically select an appropriate font, the invention instead enumerates all the fonts available in the system and organizes them by name. In the practice of the invention, and as illustrated in Figs. 4 and 5, this enumeration function is generally done at initialization time by creating a linked list of structures that contain various categories of information about the native fonts. Each font name category then contains a list of all the scripts supported, as well as the graphical font objects associated with that script. The outputting of a subject text to a screen display, printer or otherwise can therefore be performed by traversing the linked list. Figs. 4 and 5 adhere to the syntax of the C++ language, although it will be understood that other programming languages may be used.

As illustrated in Fig. 5, the CUnicodeFontManager module 516 contains a linked list of CFamilyFont modules 518, each of which contains a CUnitFontInfoManager module 522 that contains a linked list object 520 containing a sequence of CUnitFontInfo objects 524 identifying scripts associated with or

capable of expressing a given symbol. Outputting the subject text is then a matter of traversing the linked list to invoke the fonts associated with the CUnitFontInfo objects 524, to send to a display screen, printer or otherwise.

To build the linked list object 520, the following Windows™ function may be used.

Table 3

```
int EnumFontFamiliesEx(HDC,
LPLOGFONT, //pointer to logical font
FONTENUMPROC, //pointer to callback function
LPARAM, //application-supplied data
DWORD); //reserved; must be zero
```

In this environment, the callback function passed to the procedure call EnumFontFamiliesEx must have the following prototype.

Table 4

```
int CALLBACK EnumFontFamExProc(
    ENUMLOGFONTEXW *lpelfe, // pointer to logical-font data
    NEWTEXTMETRICEXW *lpntme, // pointer to physical-font data
    int FontType, // type of font
    LPARAM lParam); // application-defined data
```

Inside the callback function in Table 4 above, the linked list object 520 can be built using the data passed by the system. In the particular environment of Microsoft™ Windows™, the character set used is actually a font character set which does not necessarily match the character set or code page that is returned by the integrated character set guessing algorithm of the invention. Therefore, in one

embodiment it is necessary to map the code page to a font character set property, before doing a lookup query to the linked list object 520 of available fonts.

The code in Table 5 below is as an example under Microsoft™ Windows™ of such a code page mapping.

Table 5

BYTE MapCodePageToFontCharset(short iActiveCodePage)

```
{
    switch(iActiveCodePage)
    {
        case 874:
            Return THAI_CHARSET;
        case 932:
            Return SHIFTJIS_CHARSET;
        case 936:
            Return GB2312_CHARSET;
        case 949:
            Return HANGEUL_CHARSET;
        case 950:
            Return CHINESEBIG5_CHARSET;
        case 1250:
            Return EASTEUROPE_CHARSET;
        case 1251:
            Return RUSSIAN_CHARSET;
        case 1252:
            Return ANSI_CHARSET;
        case 1253:
            Return GREEK_CHARSET;
        case 1254:
```

```

        Return TURKISH_CHARSET;
    case 1255:
        Return HEBREW_CHARSET;
    case 1256:
        Return ARABIC_CHARSET;
    case 1257:
        Return BALTIC_CHARSET;
    default:
        break;
    }
    Return DEFAULT_CHARSET;
}

```

If a client application working in tandem with the invention to process text needs to create a graphical font object, that application will need to use a return flag when creating a logical font structure according to the invention. For further example in Windows™:

Table 6

```

BYTE FlagCharset=MapCodePageToFontCharset( iActiveCodePage);
LOGFONT lFont;
(Initialization of lFont...)
lFont.lfCharSet = FlagCharset;
CreateFontIndirect ( &lFont);
...

```

When the font has been created and selected into a Device context, it may then be necessary to query information such as ordering, ligature, existence of additional glyphs that are not encoded, and other information for output presentation. Windows™ provides the following function to access these properties:

DWORD WINAPI GetFontLanguageInfo( HDC );

Another Windows™ function which is useful in this regard is:

Table 7

DWORD WINAPI GetCharacterPlacement(  
     HDC, // handle to device context  
     LPCSTR,     //pointer to string  
     Int, // number of characters in string  
     Int, //maximum extent for display  
     LPGCP\_RESULTS, //Result buffer  
     DWORD); //placement flags

The procedure call GetCharacterPlacement can be used to query information from font tag 504 about a displayed string such as its width, ordering, glyph rendering and other information that may be used by client applications or otherwise to adjust the output document.

Another embodiment of the invention involving documents made up of sections in different languages will now be described. Using the character guessing algorithm described more fully in the aforementioned copending U.S. application serial numbers \_\_\_\_\_, the text-rendering subsystem first segments the text into multiple text runs that share the same character set or Unicode range. In general, the invention manages multipart, multilanguage documents by decomposing them into discrete segments for output.

Fig. 7 illustrates the decomposition of a multilingual text. For each segment, the invention looks up the preloaded list of system fonts to select the appropriate font. Once every segment is assigned a correct font, the entire text is displayed, as illustratively implemented in the code of the following Table 8.

Table 8

For (each segment)

```
{
    Load font into Device Context
    Display text in segment
    Unload font
}
```

An additional difficulty in a multi-part context is the increased complexity that is introduced in functions such as word wrapping, cursor movement, text highlighting and other graphically oriented operations. This is because multiple text segments must be juggled, and for each textual segment, properties such as selected font, text position (coordinate of the Xstart and Ystart for the string), caret position, and other information must be tracked.

The object CCTextRunList illustrated in Fig. 8 encapsulates a list of ccTextRun objects 524 to manage the operations of such features, including to generate a multipart linked list object 526. The following procedure call is intended to create this linked list object 526.

Table 9

```
While ( Not end of text)
{
    EvaluateTextCharset();
    Build text run with same charset...
    Add new TextRun in List(...);
}
```

The invention uses the technique described above to display the data in a multipart, multilanguage environment. In this case, the invention must likewise address complications to common word processing and other functions like text highlighting, text cursor placement, cursor movement, and so forth. The complexity for implementing these features is increased by the fact that the text is decomposed into multiple segments or runs. For example, to compute the extent of the text, in this embodiment the invention separately computes the extent of each segment and then adds them together.

Table 10

```

SIZE GetMutilingualTextExtent(HDC hdc)
{
    SIZE RetSize;
    For (each segment)
    {
        Load font into Device Context
        Compute extent for segment
        Add the size to RetSize;
        Unload font
    }
    Return RetSize;
}

```

Because more than one font is involved in the display process, the size of the text displayed in each segment can vary considerably. Therefore, a once simple operation, such as highlighting the text, can become very involved in this embodiment. The invention therefore needs to compute the smallest rectangle that contains the whole text to assist in highlighting and other functions. To improve performance, this information is stored as part of the ccTextRunList objects 524, and updated during outputting to screen, printer or otherwise.

Managing the caret position and movement also requires special attention. Again this is because the text is separated into multiple segments, and the invention thus needs a function that gets the next character. In this embodiment, the invention stores a current position pointer 528 to the current text segment as illustrated in Fig. 7, as well as a set of other pointers including the illustrated begin text pointer 530, end of segment 1 pointer 532 (first segment illustrated but begin/end pointers for other segments being contemplated), and end of text pointer 534.

Table 11

```

Short GetNextCharacter()
{
    If (end of segment)
        Select next segment
    If (no more segment)
        Return End of text
    Get next character in current segment
    Return next character;
}

```

Once the character is returned, it is straightforward to retrieve its width using the font identified for the segment. The invention in this embodiment may process as many segments are necessary to properly encode the text for output. Different scripts may appear one or more times in the aggregate document.

Another environment of the invention relating to the Lotus Domino™ Global WorkBench™ illustrated in Fig. 9. Lotus Domino™ Global WorkBench™ is a tool used for the localization of Notes™ databases and Web sites. It must correctly render the data stored in Notes™ databases, which use a Lotus-proprietary universal character set (Lotus MultiByte Character Set, or LMBCS) that is converted to Unicode for output, no matter what the language version of the operating system



used at run time. In a Notes™ environment as shown, once conversion to Unicode is carried out, output font processing may be performed as described above. Alternatively, the enumerator and other modules of the invention may be configured to directly operate on the Notes™-native character code.

The foregoing description of the system and method of the invention is illustrative, and variations in configuration and implementation will occur to persons skilled in the art. For example, while the output of characters has been described with respect to display on a computer monitor or printing, other types of output such as facsimile and others can be performed. The scope of the invention is accordingly intended to be limited only by the following claims.